中南大学

《数据结构》课程实验

实验报告

实验题目 Huffman 编码

专业班级 软件工程 2005 班

学　　号 8209200504

姓　　名 李均浩

实验成绩：

批阅教师：

2021 年 5 月 5 日

一、需求分析

1.程序任务

1、根据所提供的字母数据建立一个 Huffman 树；
2、根据生成的 Huffman 树的结构，显示输出所有字母的 Huffman 编码。
3、根据产生的 Huffman 编码，实现 Huffman 编/译码器。

2.输入以及输出的形式

图 1 程序输入输出形式

3.程序功能

对各个规定的字符进行 Huffman 编码，对英文原文进行编码或者对 Huffman 编码进行解码。实现 Huffman 编码的编码器以及解码器。

4.测试数据

(a) data structure is fantastic

预期输出：

0100101010010010100011000010110111110111110010111101101101000100111000000011000101100000100101110000101001111111

(b) I LOVE Programing

预期输出：

000100100001000011000111111010000111011101011001111011010101001101100110000111110

(c) CSU is Number ONE

预期输出：

11111110011110000100111000001000111100011010111111011101000011010001 01

(d) 10010000100001100011111101000011100011011110

预期输出：i love you

(e)

1001000100010100111111011101000011000111010000000101000011110100111010100000000110011011001101100001001000011111111110001000010010001111101011000000100010100111000000101100000000110011111001111011101111111001000001111101110000011111101110101110 1

预期输出：i never own a girl friend but i can new an object by cpp

(f) (错误输入) 45456

预期输出：未知编码阻止了解码程序的运行！（随后退出程序）

(g) (错误输入) 01011011101010101011

预期输出：未知编码阻止了解码程序的运行！（随后退出程序）

二、概要设计

1.抽象数据类型定义：

ADT Huffmantree

{

数据对象：D={ai| ai ∈Charset,i=1,2,3,,,,,n,n≥0}

数据关系：R1＝{< ai-1, ai >| ai-1, ai ∈D, i=2,3,,,,,n}

基本操作：

Initialization(&HT，&HC,w，n,ch)

操作结果：根据 n 个字符及其它们的权值 w[i],建立 Huffman 树 HT,用字符数组 ch[i]作为中间存储变量，最后字符编码存到 HC 中；

Encodeing(n)

操作结果：根据建好的 Huffman 树，对文件进行编码，编码结果存入到文件 CodeFile 中；

Decodeing(HT,n)

操作结果：根据已经编译好的包含 n 个字符的 Huffman 树 HT，将文件的代码进行翻译，结果存入文件 TextFile 中。

} ADT Huffmantree

2.主程序的流程

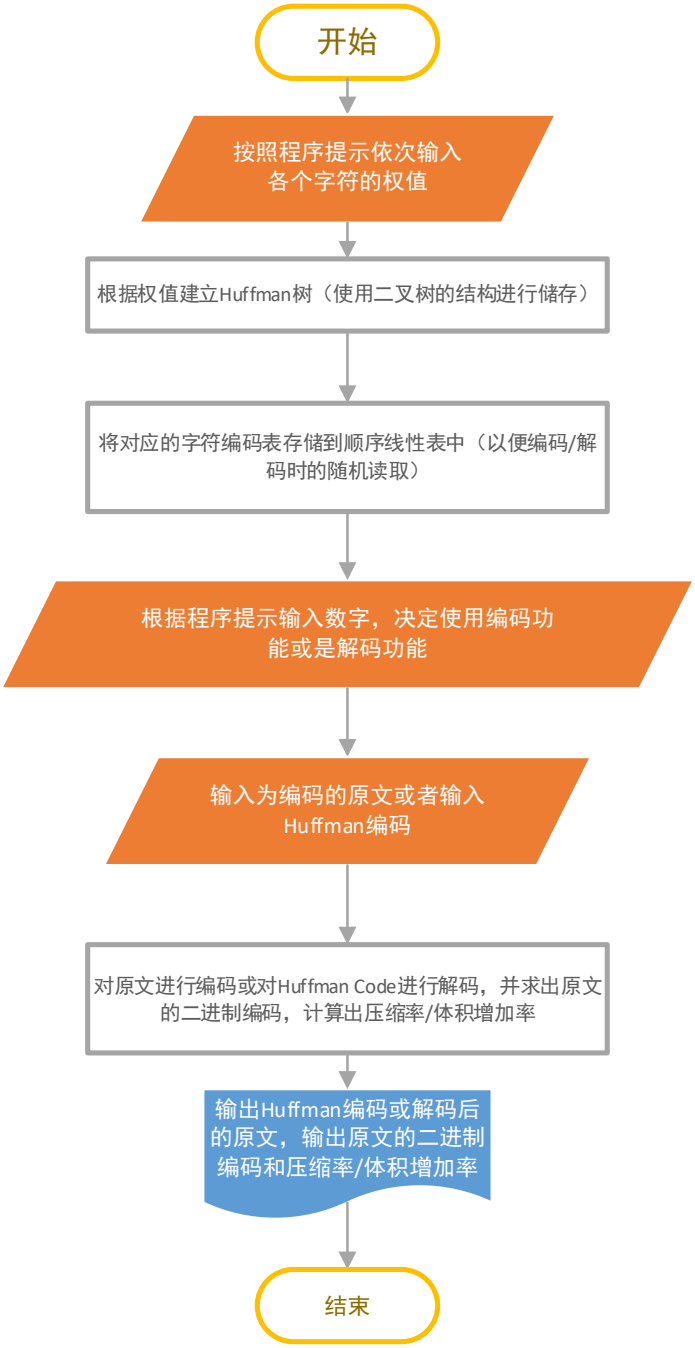

图 2 主程序的流程

三、详细设计

1.模块伪码

(1) void GetAlphabetFreq(TreeNode* node_array) 开始

```
输出<< "**********获取字母频度表**********" << endl;

auto sp = (TreeNode)malloc(sizeof(HuffmanTreeNode));
如果(sp == 空指针)
    退出程序(OVERFLOW);
sp->letter = ' ';
输出<< "请输入\' 空格 \'的权值：";
输入>> sp->weight;
sp->isDeleted = false;
sp->left = nullptr;
sp->right = nullptr;
sp->pre_order_counted_times = 0;
node_array[0] = sp;

for (int i = 0; i < 26; ++i) {
    auto tn = (TreeNode)malloc(sizeof(HuffmanTreeNode));
    if (tn == nullptr)
        exit(OVERFLOW);
    tn->letter = (char)('a' + i);
    输出 << "请输入\' " << tn->letter << " \'的权值：";
    输入 >> tn->weight;
    tn->isDeleted = false;
    tn->left = nullptr;
    tn->right = nullptr;
    tn->pre_order_counted_times = 0;
    node_array[i + 1] = tn;
}
输出 << endl << "**********获取结束**********" << endl;
array_length = 27;
```

结束

(2) TreeNode GetMinWeight(TreeNode* node_array) 开始

```
int min_index = 10000;
int min_weight = 10000;
for (int i = 0; i < array_length; ++i) {
    如果 (node_array[i]->weight < min_weight && !node_array[i]->isDeleted) {
        min_weight = node_array[i]->weight;
        min_index = i;
    }
```

```
    }
    node_array[min_index]->isDeleted = true;
    返回 node_array[min_index];
结束
```

(3)
```
bool isEmpty(TreeNode* node_array) 开始
    for (int i = 0; i < array_length; ++i)
        如果 (!node_array[i]->isDeleted)
            返回 false;
    返回 true;
结束
```

(4)
```
void SaveNodeToArray(TreeNode new_node, TreeNode* node_array) 开始
    node_array[array_length] = new_node;
    array_length++;
结束
```

(5)
```
void CreateHuffmanTree(HuffmanTree& t, TreeNode* node_array) 开始
    TreeNode parent_node;
    (无限循环) {
        TreeNode node_1, node_2;
        node_1 = GetMinWeight(node_array);
        node_2 = GetMinWeight(node_array);
        parent_node = 分配类型为(TreeNode)大小为(sizeof(HuffmanTreeNode))的内存空间;
        如果 (parent_node == 空指针)
            退出程序(OVERFLOW);
        parent_node->isDeleted = false;
        parent_node->left = node_1;
        parent_node->right = node_2;
        parent_node->letter = '#';
        parent_node->weight = node_1->weight + node_2->weight;
        parent_node->pre_order_counted_times = 0;
        如果 (isEmpty(node_array))
            退出循环;
        SaveNodeToArray(parent_node, node_array);
    }
    t.root = parent_node;
结束
```

(6) void HuffmanCodeGenerator(HuffmanTree t, string* huffman_code_list) 开始

```
string s;
s = "";
TreeNode n = t.root;
Stack node_stack;
初始化栈(node_stack);
无限循环 {
    如果 (n == t.root) {
        如果 (n->pre_order_counted_times == 0) {
            n->pre_order_counted_times++;
            Push(node_stack, n);
            n = n->left;
            s += '1';
        }
        如果 (n->pre_order_counted_times == 1) {
            n->pre_order_counted_times++;
            入栈(node_stack, n);
            n = n->right;
            s += '0';
        }
        如果 (n->pre_order_counted_times == 2)
            break;
    }
    否则 {
        如果 (n->letter == '#') {
            如果 (n->pre_order_counted_times == 0) {
                n->pre_order_counted_times++;
                入栈 (node_stack, n);
                n = n->left;
                s += '1';
            }
            如果 (n->pre_order_counted_times == 1) {
                n->pre_order_counted_times++;
                入栈 (node_stack, n);
                n = n->right;
                s += '0';
            }
            如果 (n->pre_order_counted_times == 2) {
                出栈 (node_stack, n);
                s = s.substr(0, s.length() - 1);
            }
        }
        否则 {
            std::cout << '\'' << n->letter << '\'' << "的权值为：" << n->weight
```

```
<< ",Huffman编码为：" << s << endl;
                //n->huffman_code = (string*)malloc(100);
                //*(n->huffman_code) = s;
                如果 (n->letter != ' ')
                    huffman_code_list[n->letter - 'a' + 1] = s;
                如果 (n->letter == ' ')
                    huffman_code_list[0] = s;
                否则 (node_stack, n);
                s = s.substr(0, s.length() - 1);
            }
        }
    }
结束
```

(7)
```
string SearchHuffmanCode(char c, string* huffman_code_list) 开始
    返回 huffman_code_list[c - 'a' + 1];
结束
```

(8)
```
string HuffmanEncoder(string plaintext, string* huffman_code_list) 开始
    将plaintext全部转为小写;
    string ciphertext;
    for (char i : plaintext) {
        如果 (i == ' ')
            ciphertext += huffman_code_list[0];
        否则
            ciphertext += SearchHuffmanCode(i, huffman_code_list);
    }
    返回 ciphertext;
结束
```

(9)
```
string HuffmanDecoder(const string& ciphertext, string* huffman_code_list) {
    string plaintext;
    int length = 1; //10010000100001100011111101000011100011011110
    int start_index = 0;
    bool isMatch = false;
    无限循环 {
        如果 (start_index + length > ciphertext.length())
            返回 plaintext;
        string temp = ciphertext.substr(start_index, length);
        for (int i = 0; i < 27; ++i) {
            如果 (temp == huffman_code_list[i]) {
```

```
                isMatch = true;
                如果 (i == 0)
                    plaintext += ' ';
                否则
                    plaintext += 转为(char)(i + 'a' - 1);
            }
        }
        如果 (isMatch) {
            start_index += length;
            length = 1;
            isMatch = false;
        }
        否则{
            如果 (start_index + length >= ciphertext.length()) {
                输出 << "未知编码阻止了解码程序的运行！" << endl;
                退出程序(错误代码：INVALID_INPUT);
            }
            length++;
        }
    }
结束
```

(10)
```
string GetBinCode(const string& s) 开始
    string bin_code;
    输出 << "**********原文的二进制编码**********" << endl;
    for (char i : s)
    {
        char temp[100];
        将第s的第i位转为二进制编码存入temp(i, temp, 2);
        输出 << i << '\t' << temp << endl;
        bin_code += temp;
    }
    返回 bin_code;
结束
```

2.函数调用关系图

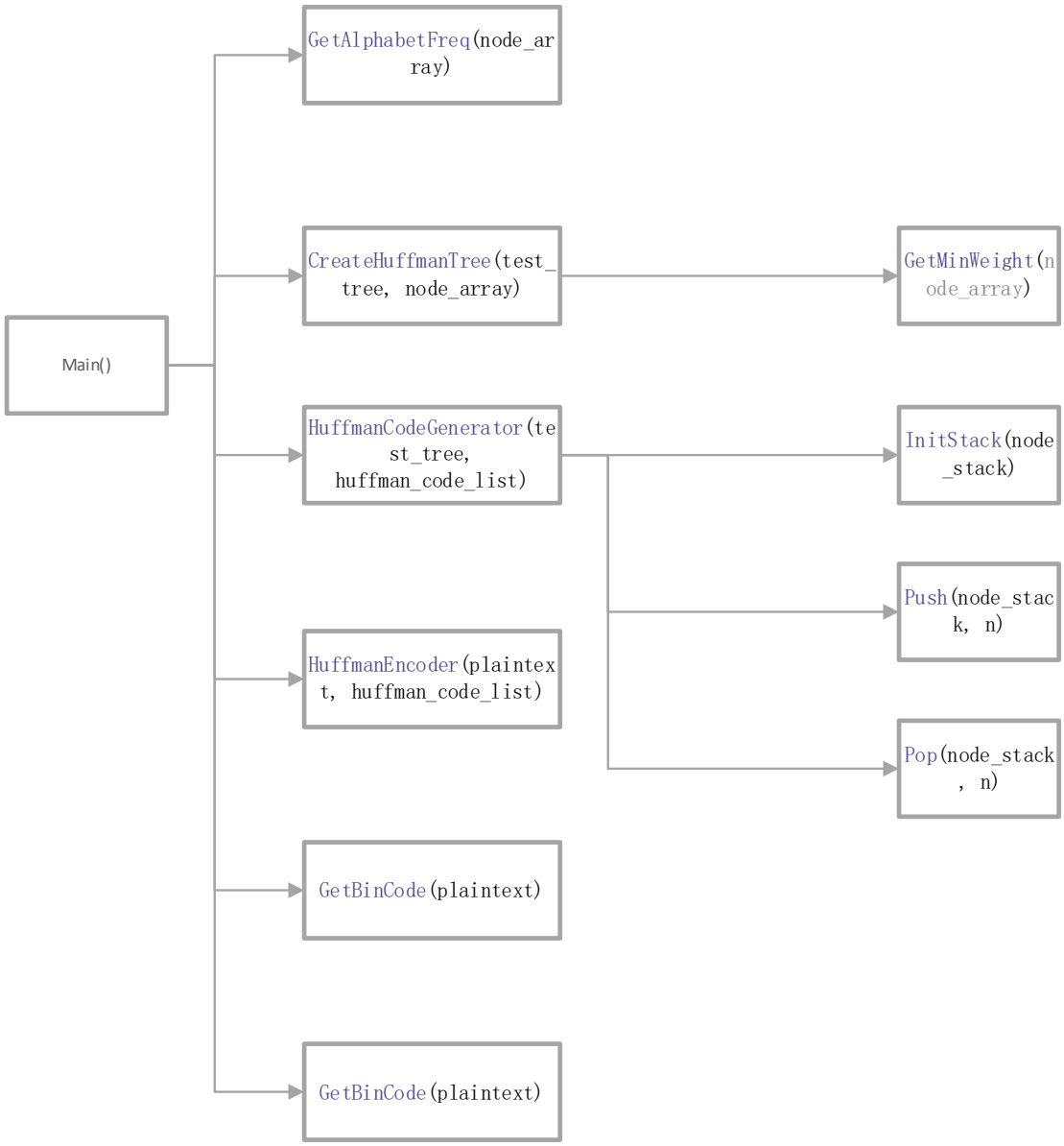

图 3 函数调用关系图

四、调试分析

1.问题复现

(1) 输入字符选择程序功能之后无法正常输入字符串

(a)错误信息

(b)错误源码

```
320     //对应地将英文转换为Huffman编码处理后的密文
321 start:
322     cout << "**********请选择编码或者解码**********\n  #键入 1 将明文转写为Huffman Code#\n  #键入 2 将Huffman Code转化为明文#" << endl;
323     char choice;
324     cin >> choice;
325
326     switch (choice) {
327     case '1': {
328         char plaintext[1000] = { 0 };
329         cout << endl << endl << "**********请输入原文**********" << endl;
330         gets_s(plaintext);
331         string ciphertext = HuffmanEncoder(plaintext, huffman_code_list);
332         cout << "Huffman Code如下：" << endl;
333         cout << ciphertext << endl;
334         cout << "**********转写结束**********" << endl << endl;
335         string bin_text = GetBinCode(plaintext);
336         cout << "**********原文的二进制编码**********" << endl;
337         cout << bin_text << endl;
338         double compression_ratio = (1.0 - (double)ciphertext.length() / (double)bin_text.length()) * 100;
339         cout << endl << "压缩率 ---> " << compression_ratio << '%' << endl;
340         break;
341     }
342     case '2': {
343         char ciphertext[1000] = { 0 };
344         cout << endl << endl << "**********请输入Huffman Code**********" << endl;
345         gets_s(ciphertext);
346         string plaintext = HuffmanDecoder(ciphertext, huffman_code_list);
347         cout << "明文如下：" << endl;
348         cout << plaintext << endl;
349         cout << "**********转写结束**********" << endl << endl << endl;
350         string bin_text = GetBinCode(plaintext);
351         cout << "**********原文的二进制编码**********" << endl;
352         cout << bin_text << endl;
353         string ciphertext_copy = ciphertext;
354         double compression_ratio = ((double)bin_text.length() / (double)ciphertext_copy.length() - 1.0) * 100;
355         cout << endl << "体积增长率 ---> " << compression_ratio << '%' << endl;
356         break;
357     }
358     default:
359         exit(_Code:INVALID_INPUT);
360     }
361     cout << "**********程序运行结束**********" << endl << endl << endl << endl << endl;
362     goto start;
```

(c)错误解释

在执行 cin >> choice 之后，缓冲区中存留了一个'\n'字符，导致程序运行至 case '1' 或 case '2' 时首先读入了'\n'导致了输入的结束，即无法正常输入字符串至编码/解码模块中。

(d)解决方案

增加三处对缓存区的处理语句，如下图所示：

```
    //对应地将英文转换为Huffman编码处理后的密文
start:
    cout << "**********请选择编码或者解码**********\n  #键入 1 将明文转写为Huffman Code#\n  #键入 2 将Huffman Code转化为明文#" << endl;
    char choice;
    fflush(stdin);
    cin >> choice;

    switch (choice) {
    case '1': {
```

```
    switch (choice) {
    case '1': {
        char plaintext[1000] = { 0 };
        cout << endl << endl << "**************请输入原文**************" << endl;
        cin.ignore();
        gets_s(&plaintext);
        string ciphertext = HuffmanEncoder(plaintext, huffman_code_list);
```

```
        cout << endl << "压缩率 ---> " << compression_ratio << '%' << endl;
        break;
    }
    case '2': {
        char ciphertext[1000] = { 0 };
        cout << endl << endl << "**********请输入Huffman Code**********" << endl;
        cin.ignore();
        gets_s(&ciphertext);
        string plaintext = HuffmanDecoder(ciphertext, huffman_code_list);
```

2. 算法的时空分析

(1)改进设想

部分判断条件分类可以合并，以减少操作的繁琐。

程序编写中有部分变量可以通过一定方式省去，能节省运行占用的空间。

3. 经验与体会

使用 Huffman 树生成的 Huffman 编码在实际应用非常广泛而且意义重大，Huffman 编码使用变长编码表对字符进行编码，它通过评估来源符号出现机率的方法得到，出现机率高的字母使用较短的编码，反之出现机率低的则使用较长的编码，这便使编码之后的字符串的平均长度、期望值降低，从而达到无损压缩数据的目的。在我编写程序的过程中能明显感觉到这一点，压缩率一般能到 30%左右，这样的话就能更大程度地使用存储资源。

五、用户使用说明

1. 根据提示输入从空格到'a'到'z'的频率。

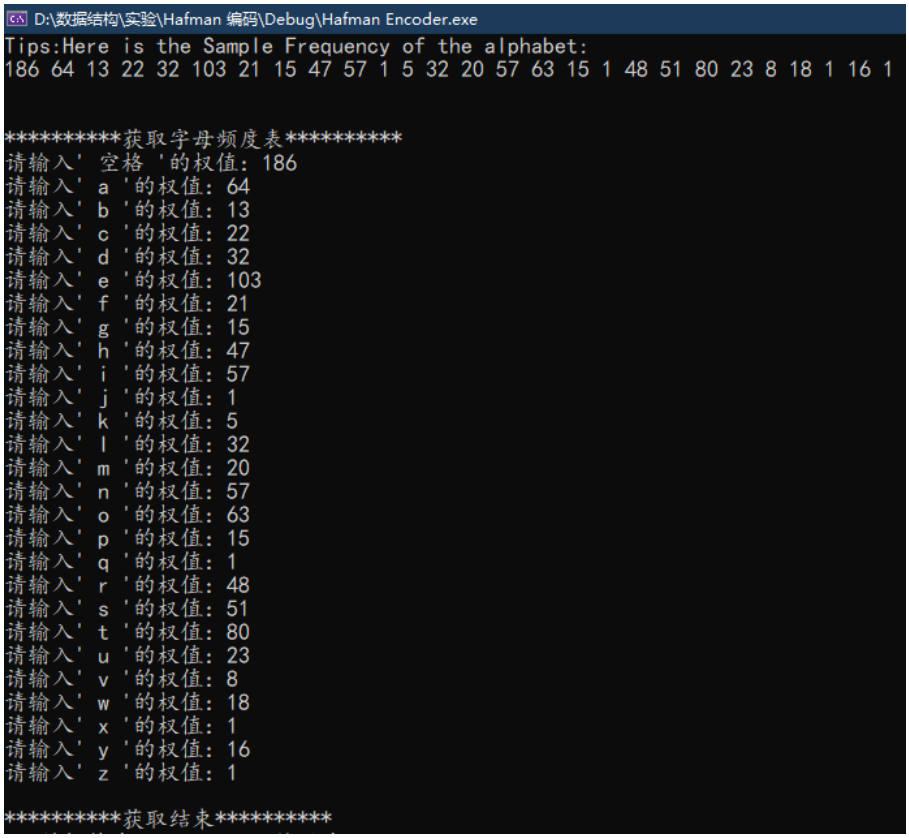

图 4.1 操作演示 1

2. 输出各个字符的对应 Huffman 编码，显示给用户。

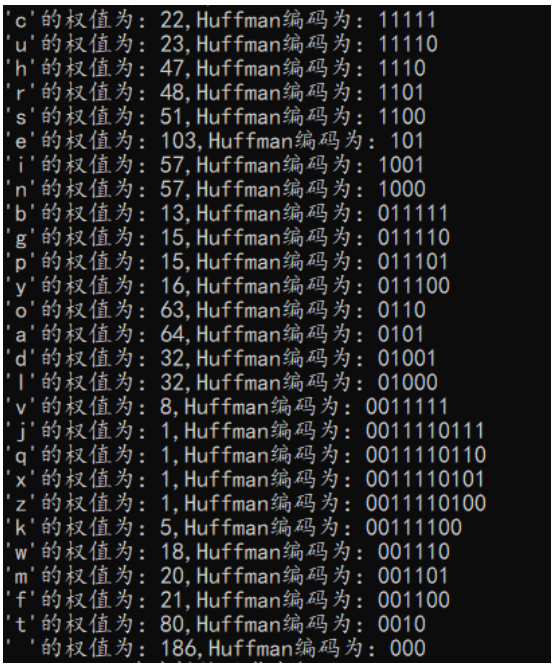

图 4.2 操作演示 2

3. 根据提示，选择进行编码或者解码。

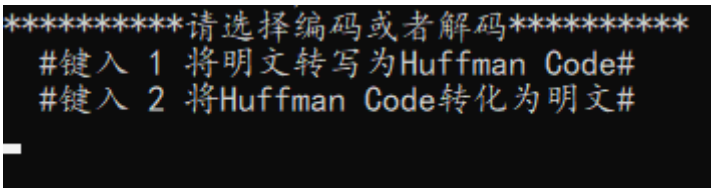

图 4.3 操作演示 3

4. 若选择了编码，则输入一段文字（允许大小写英文字母以及空格），随后输出 Huffman 编码串以及压缩率。

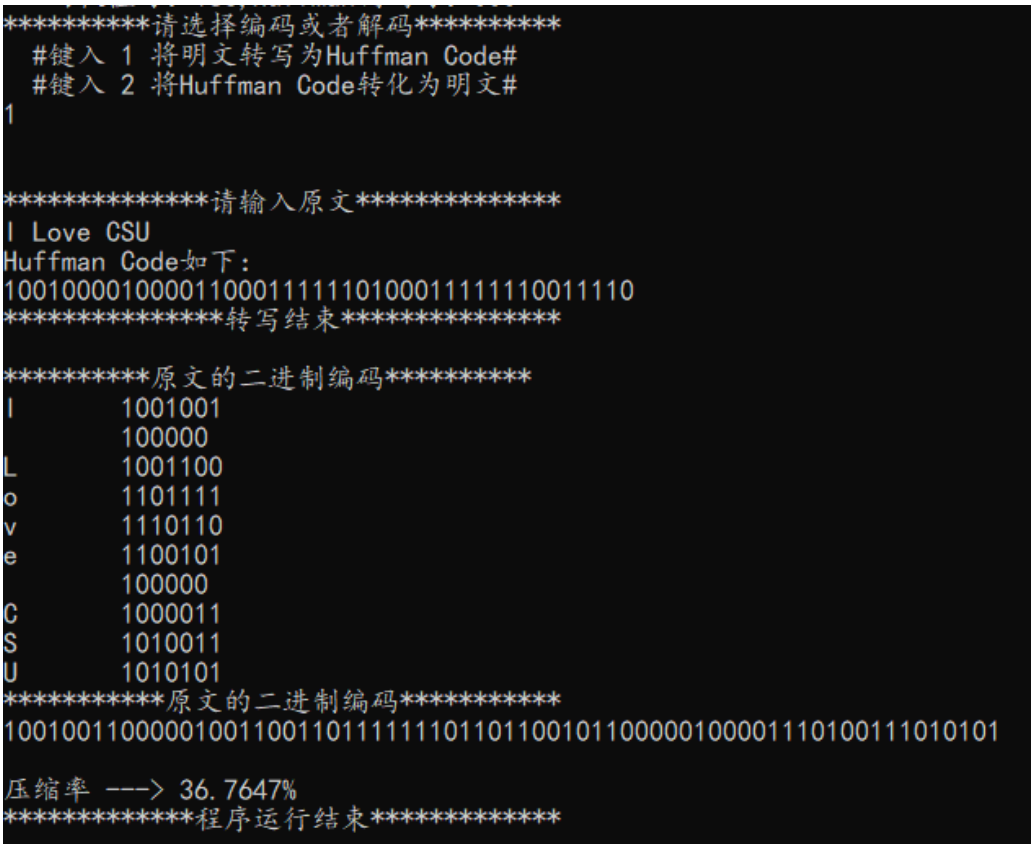

图 4.4 操作演示 4

5. 若选择了解码，则输入一段 Huffman 编码串，随后输出 Huffman 编码串对应的原文以及以及解压后的体积增加率。

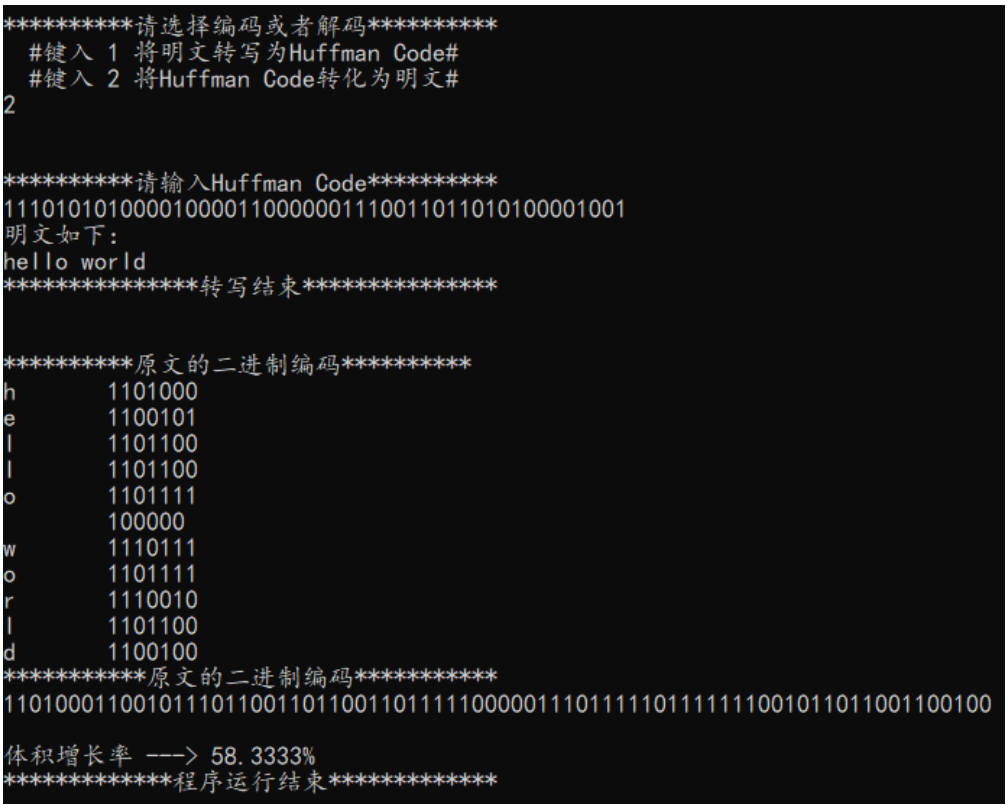

图 4.5 操作演示 5

6. 随后程序会跳转至第 3 步，供用户使用同一套字符频率产生的 Huffman 编码，进行重复编码/转码。

六、测试结果

(1)输入：（选择编码功能）data structure is fantastic

输出：

```
**********请选择编码或者解码**********
  #键入 1 将明文转写为Huffman Code#
  #键入 2 将Huffman Code转化为明文#
1


**************请输入原文**************
data structure is fantastic
Huffman Code如下:
0100101010010010100011000010110111110111110010111101101010001001110000000110001011000001001011100001010011111
****************转写结束****************

**********原文的二进制编码**********
d       1100100
a       1100001
t       1110100
a       1100001
        100000
s       1110011
t       1110100
r       1110010
u       1110101
c       1100011
t       1110100
u       1110101
r       1110010
e       1100101
        100000
i       1101001
s       1110011
        100000
f       1100110
a       1100001
n       1101110
t       1110100
a       1100001
s       1110011
t       1110100
i       1101001
c       1100011
************原文的二进制编码************
1100100110000111101001100001100000111001111101001110010111010111000111110100111010111100101100101100000110100111001110000011001101100001110111011101001100001111001111101001101001100011

压缩率 ---> 40.3226%
**************程序运行结束**************
```

(2)输入：（选择编码功能）I LOVE Programing

输出：

```
**********请选择编码或者解码**********
  #键入 1 将明文转写为Huffman Code#
  #键入 2 将Huffman Code转化为明文#
1


***************请输入原文***************
I LOVE Programing
Huffman Code如下:
1001000010000110001111110100001110111010110011111011010101001101100110000011110
****************转写结束****************

**********原文的二进制编码**********
I       1001001
        100000
L       1001100
O       1001111
V       1010110
E       1000101
        100000
P       1010000
r       1110010
o       1101111
g       1100111
r       1110010
a       1100001
m       1101101
i       1101001
n       1101110
g       1100111
************原文的二进制编码************
1001001100000100110010011111010110100010110000010100001110010110111111100111111001011000011101101110100111011101100111

压缩率 ---> 34.188%
**************程序运行结束**************
```

(3)输入：（选择编码功能）CSU is Number ONE
输出：

```
**********请选择编码或者解码**********
  #键入 1 将明文转写为Huffman Code#
  #键入 2 将Huffman Code转化为明文#
1


**************请输入原文**************
CSU is Number ONE
Huffman Code如下:
11111110011110000100111000001000111100011010111111011101000011010001 01
***************转写结束***************

**********原文的二进制编码**********
C       1000011
S       1010011
U       1010101
        100000
i       1101001
s       1110011
        100000
N       1001110
u       1110101
m       1101101
b       1100010
e       1100101
r       1110010
        100000
O       1001111
N       1001110
E       1000101
***********原文的二进制编码***********
1000011101001110101011000001101001111001110000010011101110101110110111000101100101111001010000010011111001110100010 1

压缩率 ---> 39.6552%
**************程序运行结束**************
```

(4)输入：（选择解码功能）1001000010000110001111111010000111000110111 10
输出：

```
**********请选择编码或者解码**********
  #键入 1 将明文转写为Huffman Code#
  #键入 2 将Huffman Code转化为明文#
2


**********请输入Huffman Code**********
1001000010000110001111111010000111000110111 10
明文如下:
i love you
***************转写结束***************


**********原文的二进制编码**********
i       1101001
        100000
l       1101100
o       1101111
v       1110110
e       1100101
        100000
y       1111001
o       1101111
u       1110101
***********原文的二进制编码***********
1101001100000110110011011111111011011001011000001111001110111111110101

体积增长率 ---> 54.5455%
**************程序运行结束**************
```

(5)输入：（选择解码功能）

100100010001010011111101110100001100011101000000001010000111101001110101000000001
100110110011011000010010000111111111000100001001000111110101100000010001010011 10
000010110000000011001111100111101111011111100100000111110111000001111101110101 1101

输出：

```
**********请选择编码或者解码**********
  #键入 1 将明文转写为Huffman Code#
  #键入 2 将Huffman Code转化为明文#
2


**********请输入Huffman Code**********
1001000100010100111111011101000011000111010000000010100001111010011101010000000
11110111000001111101110101 1101
明文如下:
i never own a girl friend but i can new an object by cpp
***************转写结束***************


**********原文的二进制编码**********
i       1101001
        100000
n       1101110
e       1100101
v       1110110
e       1100101
r       1110010
        100000
o       1101111
```

(6)输入：（选择解码功能）(错误输入，不合法的 Huffman 编码串) 45456

输出：

```
**********请选择编码或者解码**********
  #键入 1 将明文转写为Huffman Code#
  #键入 2 将Huffman Code转化为明文#
2


**********请输入Huffman Code**********
45456
未知编码阻止了解码程序的运行!

D:\数据结构\实验\Hafman 编码\Debug\Hafman Encoder.exe (进程 26468)已退出，代码为 1195721944。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

(7)输入：（选择解码功能）(错误输入，不存在的 Huffman 编码串) 0101101110101010 1011

输出：

```
**********请选择编码或者解码**********
  #键入 1 将明文转写为Huffman Code#
  #键入 2 将Huffman Code转化为明文#
2


**********请输入Huffman Code**********
01011011101010101011
未知编码阻止了解码程序的运行!

D:\数据结构\实验\Hafman 编码\Debug\Hafman Encoder.exe (进程 40396)已退出，代码为 1195721944。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

七、附录

```
#pragma warning (disable:4996)

#include <iostream>
#include <cstdio>
#include<cassert>
#include <string>
#include <algorithm>

#define ERROR 0
#define SUCCESS 1
#define TRUE 1
#define FALSE 0
#define STACK_INIT_SIZE 300
#define STACK_INCREMENT 10

//ERROR_EXIT_CODE
#define INVALID_INPUT 0x474544D8

//开启DEBUG输出
//#define DEBUG_MODE_ON

using namespace std;

//全局变量
int array_length;

typedef int Status;

//哈夫曼树节点
typedef struct HuffmanTreeNode {
    char letter;
    int weight;
    HuffmanTreeNode* left;
    HuffmanTreeNode* right;
    bool isDeleted;
    int pre_order_counted_times;
} *TreeNode;

//哈夫曼树
struct HuffmanTree {
    HuffmanTreeNode* root;
};
```

```
typedef TreeNode ElemType;

typedef struct SqStack {
    ElemType* base;
    ElemType* top;
    int stack_size;
} Stack;

//初始化一个栈
Status InitStack(Stack& s) {
    s.base = (ElemType*)malloc(STACK_INIT_SIZE * sizeof(ElemType));
    if (s.base == nullptr) {
        perror("Unable to allocate to memory space");
        exit(OVERFLOW);
    }
    else {
        s.top = s.base;
        s.stack_size = STACK_INIT_SIZE;
        return SUCCESS;
    }
}

//将新的元素推入栈中
Status Push(Stack& s, TreeNode e) {
    if ((s.top - s.base) >= s.stack_size) {//检查是否栈存满
        //重新追加空间，大小为STACK_INCREMENT
        s.base = (ElemType*)realloc(s.base, s.stack_size + STACK_INCREMENT);
        //检查时是否成功分配到了内存空间
        if (s.base == nullptr) {
            perror("Unable to allocate to memory space");
            exit(OVERFLOW);
        }
        //更新栈顶位置和栈大小(stack_size)记录
        s.top = s.base + s.stack_size;
        s.stack_size = s.stack_size + STACK_INCREMENT;
    }
    *s.top = e;
    s.top++;
    return SUCCESS;
}

//出栈
Status Pop(Stack& s, ElemType& e) {
```

```
    if (s.top == s.base) {
        return ERROR;
    }
    else {
        s.top--;
        e = *s.top;
        return SUCCESS;
    }
}

//判断栈是否为空
Status StackEmpty(Stack s) {
    if (s.base == s.top)
        return TRUE;
    else
        return FALSE;
}

//将字母频度表存入二叉树节点，并将二叉树节点的地址存入数组中
void GetAlphabetFreq(TreeNode* node_array) {
    cout << "**********获取字母频度表**********" << endl;

    auto sp = (TreeNode)malloc(sizeof(HuffmanTreeNode));
    if (sp == nullptr)
        exit(OVERFLOW);
    sp->letter = ' ';
    cout << "请输入\' 空格 \'的权值：";
    cin >> sp->weight;
    sp->isDeleted = false;
    sp->left = nullptr;
    sp->right = nullptr;
    sp->pre_order_counted_times = 0;
    node_array[0] = sp;

    for (int i = 0; i < 26; ++i) {
        auto tn = (TreeNode)malloc(sizeof(HuffmanTreeNode));
        if (tn == nullptr)
            exit(OVERFLOW);
        tn->letter = (char)('a' + i);
        cout << "请输入\' " << tn->letter << " \'的权值：";
        cin >> tn->weight;
        tn->isDeleted = false;
        tn->left = nullptr;
        tn->right = nullptr;
```

```
            tn->pre_order_counted_times = 0;
            node_array[i + 1] = tn;
        }
        cout << endl << "**********获取结束**********" << endl;
        array_length = 27;
}

TreeNode GetMinWeight(TreeNode* node_array) {
        int min_index = 10000;
        int min_weight = 10000;
        for (int i = 0; i < array_length; ++i) {
            if (node_array[i]->weight < min_weight && !node_array[i]->isDeleted) {
                min_weight = node_array[i]->weight;
                min_index = i;
            }
        }
        node_array[min_index]->isDeleted = true;
        return node_array[min_index];
}

bool isEmpty(TreeNode* node_array) {
        for (int i = 0; i < array_length; ++i)
            if (!node_array[i]->isDeleted)
                return false;
        return true;
}

void SaveNodeToArray(TreeNode new_node, TreeNode* node_array) {
        node_array[array_length] = new_node;
        array_length++;
}

void CreateHuffmanTree(HuffmanTree& t, TreeNode* node_array) {
        TreeNode parent_node;
        while (true) {
            TreeNode node_1, node_2;
            node_1 = GetMinWeight(node_array);
            node_2 = GetMinWeight(node_array);
            parent_node = (TreeNode)malloc(sizeof(HuffmanTreeNode));
            if (parent_node == nullptr)
                exit(OVERFLOW);
            parent_node->isDeleted = false;
            parent_node->left = node_1;
            parent_node->right = node_2;
```

```
        parent_node->letter = '#';
        parent_node->weight = node_1->weight + node_2->weight;
        parent_node->pre_order_counted_times = 0;
        if (isEmpty(node_array))
            break;
        SaveNodeToArray(parent_node, node_array);
    }
    t.root = parent_node;
}

//产生Huffman编码
void HuffmanCodeGenerator(HuffmanTree t, string* huffman_code_list) {
    string s;
    s = "";
    TreeNode n = t.root;
    Stack node_stack;
    InitStack(node_stack);
    while (true) {
        if (n == t.root) {
            if (n->pre_order_counted_times == 0) {
                n->pre_order_counted_times++;
                Push(node_stack, n);
                n = n->left;
                s += '1';
            }
            if (n->pre_order_counted_times == 1) {
                n->pre_order_counted_times++;
                Push(node_stack, n);
                n = n->right;
                s += '0';
            }
            if (n->pre_order_counted_times == 2)
                break;
        }
        else {
            if (n->letter == '#') {
                if (n->pre_order_counted_times == 0) {
                    n->pre_order_counted_times++;
                    Push(node_stack, n);
                    n = n->left;
                    s += '1';
                }
                if (n->pre_order_counted_times == 1) {
                    n->pre_order_counted_times++;
```

```
                    Push(node_stack, n);
                    n = n->right;
                    s += '0';
                }
                if (n->pre_order_counted_times == 2) {
                    Pop(node_stack, n);
                    s = s.substr(0, s.length() - 1);
                }
            }
            else {
                std::cout << '\'' << n->letter << '\'' << "的权值为：" <<
n->weight << ",Huffman编码为：" << s << endl;
                if (n->letter != ' ')
                    huffman_code_list[n->letter - 'a' + 1] = s;
                if (n->letter == ' ')
                    huffman_code_list[0] = s;
                Pop(node_stack, n);
                s = s.substr(0, s.length() - 1);
            }
        }
    }
}

string SearchHuffmanCode(char c, string* huffman_code_list) {
    return huffman_code_list[c - 'a' + 1];
}

string HuffmanEncoder(string plaintext, string* huffman_code_list) {
    transform(plaintext.begin(), plaintext.end(), plaintext.begin(), ::tolower);
    string ciphertext;
    for (char i : plaintext) {
        if (i == ' ')
            ciphertext += huffman_code_list[0];
        else
            ciphertext += SearchHuffmanCode(i, huffman_code_list);
    }
    return ciphertext;
}

string HuffmanDecoder(const string& ciphertext, string* huffman_code_list) {
    string plaintext;
    int length = 1; //10010000100001100011111110100001110001101111O
    int start_index = 0;
    bool isMatch = false;
```

```
    while (true) {
        if (start_index + length > ciphertext.length())
            return plaintext;
        string temp = ciphertext.substr(start_index, length);
        for (int i = 0; i < 27; ++i) {
            if (temp == huffman_code_list[i]) {
                isMatch = true;
                if (i == 0)
                    plaintext += ' ';
                else
                    plaintext += (char)(i + 'a' - 1);
            }
        }
        if (isMatch) {
            start_index += length;
            length = 1;
            isMatch = false;
        }
        else {
            if (start_index + length >= ciphertext.length()) {
                cout << "未知编码阻止了解码程序的运行！" << endl;
                exit(INVALID_INPUT);
            }
            length++;
        }
    }
}

string GetBinCode(const string& s)
{
    string bin_code;
    cout << "**********原文的二进制编码**********" << endl;
    for (char i : s)
    {
        char temp[100];
        itoa(i, temp, 2);
        cout << i << '\t' << temp << endl;
        bin_code += temp;
    }
    return bin_code;
}

int main() {
    cout << "Tips:Here is the Sample Frequency of the alphabet:" << endl
```

```
        << "186 64 13 22 32 103 21 15 47 57 1 5 32 20 57 63 15 1 48 51 80 23 8 18 1
16 1" << endl << endl << endl;
    //创建Huffman树
    auto* node_array = new TreeNode[1000];
    GetAlphabetFreq(node_array);
    HuffmanTree test_tree{};
    CreateHuffmanTree(test_tree, node_array);
    auto* huffman_code_list = new string[30];

    //获取27种字符的Huffman编码
    HuffmanCodeGenerator(test_tree, huffman_code_list);

    //对应地将英文转换为Huffman编码处理后的密文
start:
    cout << "**********请选择编码或者解码**********\n  #键入 1 将明文转写为Huffman
Code#\n  #键入 2 将Huffman Code转化为明文#" << endl;
    char choice;
    fflush(stdin);
    cin >> choice;

    switch (choice) {
    case '1': {
        char plaintext[1000] = { 0 };
        cout << endl << endl << "**************请输入原文**************" << endl;
        cin.ignore();
        gets_s(plaintext);
        string ciphertext = HuffmanEncoder(plaintext, huffman_code_list);
        cout << "Huffman Code如下：" << endl;
        cout << ciphertext << endl;
        cout << "***************转写结束***************" << endl << endl;
        string bin_text = GetBinCode(plaintext);
        cout << "***********原文的二进制编码***********" << endl;
        cout << bin_text << endl;
        double compression_ratio = (1.0 - (double)ciphertext.length() /
(double)bin_text.length()) * 100;
        cout << endl << "压缩率 ---> " << compression_ratio << '%' << endl;
        break;
    }
    case '2': {
        char ciphertext[1000] = { 0 };
        cout << endl << endl << "**********请输入Huffman Code**********" << endl;
        cin.ignore();
        gets_s(ciphertext);
        string plaintext = HuffmanDecoder(ciphertext, huffman_code_list);
```

```
            cout << "明文如下：" << endl;
            cout << plaintext << endl;
            cout << "***************转写结束***************" << endl << endl << endl;
            string bin_text = GetBinCode(plaintext);
            cout << "***********原文的二进制编码***********" << endl;
            cout << bin_text << endl;
            string ciphertext_copy = ciphertext;
            double compression_ratio = ((double)bin_text.length() /
(double)ciphertext_copy.length() - 1.0) * 100;
            cout << endl << "体积增长率 ---> " << compression_ratio << '%' << endl;
            break;
        }
        default:
            exit(INVALID_INPUT);
        }
        cout << "*************程序运行结束*************" << endl << endl << endl <<
endl << endl;
        goto start;
}
```

源代码 1 Huffman Encoder and Decoder.cpp